ซอฟต์แวร์

สิทธิศักดิ์ คล่องดี

# ASSEMBLY ใครว่ายาก

***ศึกษาการเขียนภาษาแอสเซมบลีอย่างถูกขั้นตอน แล้วท่านจะรู้ว่าภาษาแอสเซมบลีมันไม่ยากอย่างที่คิด***

ภาษาแอสเซมบลีจัดว่าเป็นภาษาที่เก่าแก่ที่สุดก็ว่าได้ เพราะอยู่คู่กับคอมพิวเตอร์เรามานานแสนนานแล้ว ตั้งแต่สมัยเริ่มวิวัฒนาการทางภาษาเป็นครั้งแรก ถ้าหากกล่าวถึงภาษานี้แล้วนักคอมพิวเตอร์ท่านใดไม่รู้จักจัดว่าอยู่คนละโลกกับคอมพิวเตอร์ก็ว่าได้ ถ้าถามว่าคุณเขียนโปรแกรมภาษาแอสเซมบลีเป็นหรือไม่ ส่วนใหญ่แล้วจะได้รับคำตอบอยู่สองอย่างคือ เขียนเป็นบ้างเล็กน้อย หรือไม่ก็ตอบว่าไม่เป็นเลย เพราะมันยากเกินไป มีน้อยมากที่จะตอบว่าเขียนเป็น

ฟังดูแล้วก็ไม่น่าแปลกใจอะไรหรอกกับการที่จะตอบว่ามันยาก เพราะว่าถ้าหากเรายังคิดว่าคอมพิวเตอร์มันฉลาดอยู่ก็จะบอกว่าภาษานี้มันยาก แต่ในความเป็นจริงแล้วภาษานี้จัดว่าง่ายกว่าภาษาอื่น ๆ ถ้าหากว่าท่านได้ศึกษามันอย่างถูกต้องตามขั้นตอน เพราะภาษานี้ถ้าหากว่าท่านศึกษาผิดขั้นตอนแล้วจะทำให้ท่านไม่เข้าใจมันเอาซะเลย เพราะลักษณะการเขียนโปรแกรมภาษาแอสเซมบลีเป็นการเขียนโปรแกรมแบบตรงไปตรงมา ลักษณะของโครงสร้างทางภาษา เป็นภาษาที่มีความละเอียดอ่อนและก็ต้องใช้ความเข้าใจในภาษาเป็นอย่างสูงในการที่จะเขียนโปรแกรมออกมาได้ แต่ถ้าท่านสามารถเขียนเป็นแล้วท่านก็จะบอกว่า "หมู ๆ"

อัศวินขี่ม้าขาวเข้ามาช่วยในการเขียนโปรแกรมก็คือ ภาษาแอสเซมบลี เป็นคำพูดที่นักเขียนโปรแกรมชอบพูดกัน สิ่งนี้เป็นจริงหรือ ท่านลองพิจารณาเหตุการณ์ดังนี้

ขณะที่ท่านกำลังใช้ภาษาปาสคาลอยู่ ท่านได้รับคำสั่งให้เขียนโปรแกรมรับค่าตัวเลขโดยมีข้อกำหนดว่าโปรแกรมจะรับเฉพาะตัวเลขเท่านั้น ถ้าหากว่าไม่ใช่ตัวเลขโปรแกรมจะไม่รับและไม่แสดงค่าที่ป้อนเข้าไปนั้นออกมาที่จอภาพด้วย ถ้าจะมาดูคำสั่งในการรับค่าตัวเลขของภาษาปาสคาลก็คงมีแต่คำสั่ง READ กับ READLN เท่านั้น ซึ่งคำสั่งทั้งสองนี้จะแสดงค่าที่ป้อนเข้ามาทั้งหมดโดยไม่สนใจว่าเป็นตัวเลขหรือตัวอักษร ซึ่งแน่นอนที่สุดก็คือใช้ไม่ได้ทั้งสองคำสั่ง บางท่านอาจจะบอกว่าไม่เห็นจะยาก ก็สร้างคำสั่งขึ้นมาใหม่สิ ถูกต้องแล้ว การสร้างคำสั่งขึ้นใหม่นี้ต้องเชื่อมโยงกับแอสเซมบลีโดยตรง นอกจากนี้ยังมีอีกหลายกรณีที่ไม่ได้กล่าวถึง มาถึงตรงนี้ท่านจะยอมรับได้หรือยังครับว่าแอสเซมบลีคืออัศวินขี่ม้าขาวมาช่วยท่านในยามที่ท่านคับขันจริง ๆ และภาษานี้มันจะยังคงอยู่ต่อไปควบคู่กับเราอีกนานแสนนานตราบเท่าที่เรายังใช้คอมพิวเตอร์อยู่

สำหรับท่านผู้อ่านท่านใดที่คิดว่าภาษาแอสเซมบลียากไม่ควรจะศึกษาแล้วละก็ ท่านลองอ่านบทความนี้ดูนะครับแล้วท่านจะได้รู้ว่าภาษาแอสเซมบลีมันไม่ยากอย่างที่ท่านคิดเลย โดยผมจะแนะนำวิธีการเขียนโดยเริ่มจากง่ายไปหายาก จะเรียงเป็นขั้นตอนไว้ให้ผู้อ่านได้ศึกษาเป็นขั้นตอน พร้อมกับคำอธิบายในแต่ละขั้นตอนอย่างละเอียด

ก่อนที่จะทำการศึกษาภาษาแอสเซมบลีเราควรศึกษาตัวรีจิสเตอร์ต่าง ๆ ก่อน ว่าแต่ละตัวนั้นมีหน้าที่หลักใช้ทำอะไร (อ้างอิงจาก MicroProcessor เบอร์ 8088 ของ บริษัท Intel) โดยจะแยกออกเป็นกลุ่ม ๆ

## กลุ่มที่ 1

เราเรียกว่า DATA REGISTER ซึ่งจะประกอบไปด้วย Register ต่าง ๆ ดังต่อไปนี้คือ AX (Accumulator Register), BX (Base Register), CX (Counter Register), DX (Data Register) ซึ่งเป็นรีจิสเตอร์ 16 บิต นอกจากนี้เรายังสามารถแบ่งรีจิสเตอร์16 บิตกลุ่มนี้ออกมาใช้งานทีละ 8 บิต ได้ โดยแบ่งออกเป็น 8 บติบน และ 8 บิตล่าง เมื่อทำการแบ่งแล้วจะได้ดังนี้คือ

AX แบ่งออกเป็น AH,AL

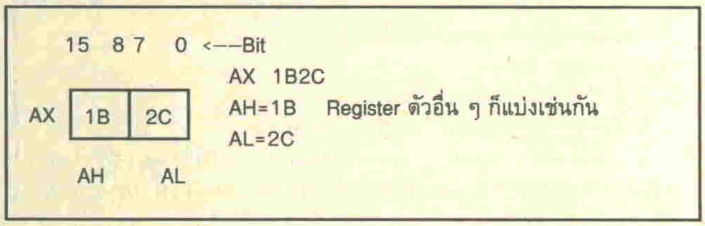
BX แบ่งออกเป็น BH,BL

CX แบ่งออกเป็น CH,CL

DX แบ่งออกเป็น DH,DL

เมื่อแบ่งแล้วเราจะเรียกเป็น High กับ Low โดย High จะใช้ตั้งแต่บิตที่ 8-15 Low จะใช้ตั้งแต่บิตที่ 0-7 ดังตัวอย่างที่ 1

**ตัวอย่างที่ 1**




## หน้าที่ของรีจิสเตอร์แต่ละตัว

AX จะใช้ในการคำนวณ เช่น การคูณและการหาร นอกจากนี้ยังใช้ในการส่ง Service Call ของอินเตอร์รัพต์แต่ละเบอร์

BX เป็นรีจิสเตอร์พื้นฐานที่ใช้จัดการกับหน่วยความจำ

CX ใช้นับจำนวนรอบในการทำงาน

DX ใช้ในการรับส่งข้อมูล

## กลุ่มที่ 2

เป็นรีจิสเตอร์แบบ 16 บิต จะแบ่งใช้ทีละ 8 บิต ไม่ได้เราเรียกว่า Pointer Register ซึ่งจะประกอบไปด้วย BP (Base Pointer), SP (Stack Pointer), SI (Source Index), DI (Destination Index) Register กลุ่มนี้จะใช้ในการจัดการกับหน่วยความจำเป็นส่วนใหญ่

## กลุ่มที่ 3

เราเรียกว่า Segment Register ซึ่งจะประกอบด้วย CS (code Segment), DS (Data Segment), SS (Stack Segment), ES (Extra Segment) เป็นรีจิสเตอร์แบบ 16 บิตทั้งหมด

CS ใช้เก็บชุดคำสั่งในการทำงาน

DS ใช้เก็บข้อมูลในการทำงาน

SS ใช้เก็บ Stack ของโปรแกรม

ES ใช้ทำงานร่วมกับเซกเมนต์อื่น ๆ

## กลุ่มที่ 4

มีตัวเดียวคือ IP (Instruction Pointer) เป็นรีจิสเตอร์ 16 บิต ใช้ในการชี้คำสั่งในการทำงาน

## กลุ่มที่ 5

มีตัวเดียวเช่นกัน คือ Status Flag ใช้ในการบอกผลลัพธ์ในการปฏิบัติงาน

การเริ่มต้นทำการศึกษาภาษาแอสเซมบลีให้ถูกวิธีนั้นเราควรเริ่มต้นที่การเขียนโปรแกรมเล็ก ๆ ในดีบัก ซึ่งเป็นโปรแกรมที่แนบติดมากับดอส ซึ่งก็คือ FILE DEBUG.COM และ File DEBUG.EXE ในดอสตั้งแต่เวอร์ชัน 5 ขึ้นไป การเริ่มใช้ดีบักเราทำได้โดยการจัดหาไฟล์ดีบักมาให้ตรงกันกับเวอร์ชันของดอสที่ท่านกำลังใช้อยู่แล้วพิมพ์

```
C:>DEBUG <Enter>
-_ <--Debug Prompt
```

การเขียนโปรแกรมบนดีบักนั้นเราไม่สามารถเขียนโปรแกรมได้ขนาดใหญ่นัก(ประมาณ 64 กิโลไบต์ หรือ 1 เซกเมนต์) แต่ในความเป็นจริงแล้วไม่ถึงเพราะต้องเสียหน่วยความจำบางส่วนของเซกเมนต์นั้นเพื่อไปใช้เป็น Program Segment Prefix(PSP) และ Stack ของตัวโปรแกรมเอง เพราะอะไรนะหรือเอาไว้เราค่อยพูดกันทีหลัง และการเขียนโปรแกรมเราจะเริ่มกันที่ Offset ที่ 100H เสมอ (ในดีบักตัวเลขทุกตัวจะเป็นเลขฐาน 16) เพราะ Offset Address ตั้งแต่ 0000-00FF เป็นส่วนของ PSP (คำสั่งในดีบักที่ผมไม่ได้อธิบายหาอ่านได้จากหนังสือคู่มือการใช้ดอส) เรามาเริ่มทำการศึกษาคำสั่งของแอสเซมบลีกันเถอะ

# คำสั่งของแอสเซมบลี

## คำสั่ง MOVE(MOV)

เป็นคำสั่งในการเคลื่อนย้ายข้อมูล และเป็นคำสั่งแรกที่เราควรจะเริ่มทำการศึกษา

**ตัวอย่างที่ 2 คำสั่ง MOVE(MOV)**

```
รูปแบบการใช้คำสั่ง  MOV Destination,Source
การทำงาน            Destination<--Source
        Destination อาจเป็น ตัวแปรหรือ REGISTER ก็ได้
        Source  อาจเป็น ตัวแปร, Register, ค่าในหน่วยความจำ
        หรือ ค่าคงที่
ตัวอย่างเช่น
    -a100  <-- การเขียนโปรแกรมจะเริ่มต้นที่ Offset 100h โดยใช้
                คำสั่ง Assemble(a,A)
    21E3:0100 mov ax,1234   ; นำค่า 1234H ไปเก็บไว้ใน AX
    21E3:0103 mov bh,14     ; นำค่า 14H  ไปเก็บไวใน BH
    21E3:0105 mov bl,fb     ; นำค่า 0FBH  ไปเก็บไว้ใน BL
    21E3:0107
        |    |
        |    |------ Offset Address
        |
        |------ Segment
```

## คำสั่ง ADD

เป็นคำสั่งที่ใช้ในการบวกเลข

**ตัวอย่างที่ 3 คำสั่ง ADD**

```
รูปแบบการใช้งาน  ADD Destination,Source
การทำงาน         Destination<-Destination+Source
        Destination อาจเป็น ตัวแปรหรือ REGISTER ก็ได้
        Source  อาจเป็น ตัวแปร, Register, ค่าในหน่วยความจำ
        หรือ ค่าคงที่
ตัวอย่างเช่น
        -a100
        21E3:0100 mov ax,15 ; AX <- 15H
        21E3:0103 add ax,4b ; AX <- AX+4BH
        21E3:0106 add ax,1f ; AX <- AX+4BH+1FH คือ คำตอบ
        21E3:0109
```

## คำสั่ง SUB

เป็นคำสั่งที่ใช้ในการลบเลข

**ตัวอย่างที่ 4 คำสั่ง SUB**

```
รูปแบบการใช้งาน        SUB Destination,Source
การทำงาน        Destination<-Destination-Source
        Destination อาจเป็น ตัวแปรหรือ REGISTER ก็ได้
        Source  อาจเป็น ตัวแปร, Register, ค่าในหน่วยความจำ
        หรือ ค่าคงที่
ตัวอย่างเช่น
        -a100
        21E3:0100 mov ah,45 ; AH<- 45H
        21E3:0102 sub ah,5  ; AH<- 45H-5H
        21E3:0105 sub ah,b  ; AH<- 45H-5H-0BH คือ คำตอบ
        21E3:0108
```

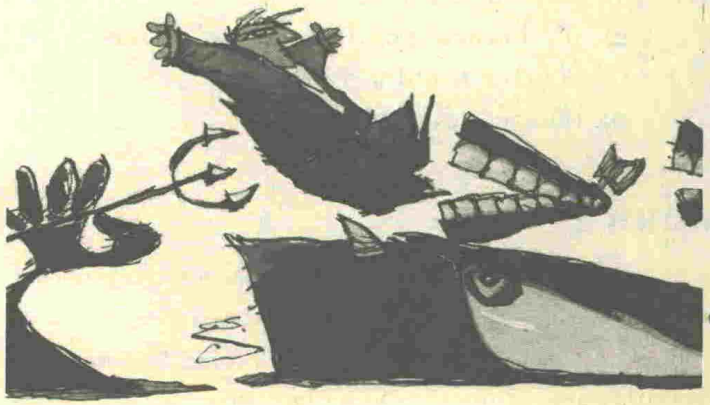
## คำสั่ง MUL

เป็นคำสั่งที่ใช้ในการคูณเลข ในที่นี้จะแบ่งออกเป็น 2 อย่าง คือ

1. การคูณเลขแบบ 8 บิต
2. การคูณเลขแบบ 16 บิต

**ตัวอย่างที่ 5 คำสั่ง MUL**

```
การคูณแบบ 8 บิต
        - ตัวตั้งจะอยู่ใน Register AL เสมอ
        - ตัวคูณจะอยู่ใน Register 8 บิต
        - ผลลัพธ์จะอยู่ใน Register AX เสมอ
การคูณแบบ 16 บิต
        - ตัวตั้งจะอยู่ใน Register AX เสมอ
        - ตัวคูณจะอยู่ใน Register 16 บิต
        - ผลลัพธ์จะอยู่ใน Register DX:AX เสมอ (Word บนจะอยู่ใน
        Register DX และ Word ล่างจะอยู่ใน Register AX)
ตัวอย่างเช่น
        จงหาคำนิพจน์ ((18H*17H)+40H)-32H
        -a100  ┌──────────── Address เริ่มต้นของรหัสคำสั่ง
        21E3:0100 mov al,18
        21E3:0102 mov bl,17
        21E3:0104 mul bl    ;AX <== AL/BL
        21E3:0106 add ax,40  ;AX <== (AL/BL)+40H
        21E3:0109 sub ax,32  ;AX <== ((AL/BL)+40H)-32H
        21E3:010C
                └──────────── Address สุดท้าย
ถ้าท่านอยากจะรู้ว่าผลลัพธ์ได้เท่าไร่ให้ใช้คำสั่ง
-g=100 10c <Enter>
    |    |
    |    |── Address สุดท้าย
    |────── Address เริ่มต้นของรหัสคำสั่ง
    ┌──────── ผลลัพธ์    ┌── ถ้าคูณแบบ 16 Bit ให้ดูค่า DX ด้วย
AX=0236  BX=0017  CX=000C  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=21E3  ES=21E3  SS=21E3  CS=21E3  IP=010C   NV UP EI
PL NZ NA PE NC
21E3:010C C3          RET
```



## คำสั่ง DIV

เป็นคำสั่งที่ใช้ในการหารเลข ในที่นี้จะแบ่งออกเป็น 2 อย่าง คือ

1. การหารเลขแบบ 8 บิต
2. การหารเลขแบบ 16 บิต

**ตัวอย่างที่ 6 คำสั่ง DIV**

```
การหารแบบ 8 บิต
        - ตัวตั้งจะอยู่ใน Register AX เสมอ
        - ตัวหารจะอยู่ใน Register 8 บิต
        - ผลลัพธ์จะอยู่ใน Register AL
        - เศษอยู่ใน Register AH
        - การหารแบบ 16 บิต
        - ตัวตั้งจะอยู่ใน Register DX:AX เสมอ
        - ตัวหารจะอยู่ใน Register 16 บิต
        - ผลลัพธ์จะอยู่ใน Register AX
        เศษอยู่ใน Register DX
ตัวอย่างเช่น จงหาค่านิพจน์ (40H-12H)/10H
        -a100
        21E3:0100 mov ax,40 ; AX <- 40H
        21E3:0103 sub ax,12 ; AX <- AX-12H
        21E3:0106 mov cl,10
        21E3:0108 div cl    ; AX <- AX/10H
        21E3:010A           ; AH=เศษ  AL=ผลลัพธ์
```

**หมายเหตุ** ในการคูณหรือหารและจะเลือกกระทำแบบ 8 บิต หรือ 16 บิต ต้องพิจารณาจากผลลัพธ์ที่จะเกิดขึ้น เช่น ถ้าตัวตั้งหรือตัวคูณมีค่ามากกว่า 256 ก็ต้องใช้วิธีการคูณแบบ 16 บิต

เป็นไปได้ว่าท่านอยากจะติดตามการทำงานของโปรแกรมไปทีละขั้นตอน (Dedug Program) ท่านสามารถใช้คำสั่งดังตัวอย่างที่ 7

**ตัวอย่างที่ 7**

```
-t=100    <Enter>
 |  |  └──── ส่วนของการ Increment ถ้าไม่กำหนดถือว่าเท่ากับ 1
 |  └──────── Address เริ่มต้นของรหัสคำสั่ง
 └──────────── Trace Command(T)
AX=0040  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=21E3  ES=21E3  SS=21E3  CS=21E3  IP=0103   NV UP EI
PL NZ NA PO NC
21E3:0103 2D1200      SUB    AX,0012 <- รหัสคำสั่งที่จะทำงาน
ต่อไป
-t <Enter>
AX=002E  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=21E3  ES=21E3  SS=21E3  CS=21E3  IP=0106   NV UP EI
PL NZ AC PE NC
21E3:0106 B110        MOV    CL,10
```

→

**ตัวอย่างที่ 7 (ต่อ)**

```
-t <Enter>
AX=002E  BX=0000  CX=0010  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=21E3  ES=21E3  SS=21E3  CS=21E3  IP=0108   NV UP EI
PL NZ AC PE NC
21E3:0108 F6F1        DIV    CL
-t <Enter>
     ┌──────── เศษ
     │  ┌──── ผลลัพธ์
AX=0E 02  BX=0000  CX=0010  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=21E3  ES=21E3  SS=21E3  CS=21E3  IP=010A   NV UP EI
NG NZ NA PO CY
21E3:010A 1F          POP    DS
```

■

เมื่อมาถึงตรงจุดนี้เป็นไปได้ว่าท่านอยากจะเก็บโปรแกรมที่เขียนขึ้นมาไว้ในดิสค์ สามารถทำได้ดังตัวอย่างที่ 8

**ตัวอย่างที่ 8**

```
-N test.bin
 |  └─ ชื่อไฟล์
 └──── คำสั่งในการกำหนดชื่อไฟล์ใน Debug
-Rbx
 |  └── Register
 └──── คำสั่งขอดูค่าในตัว Register(Register Command,R)
BX 0000 <- ค่า BX ต้องเป็น 0 ถ้าไม่เท่ากับ 0 ให้พิมพ์ 0 ลงไป
แล้วกด Enter
:_
-Rcx
CX 0000
:8 <- จำนวน Byte ที่จะเขียน หาได้จาก Address สุดท้าย ลบด้วย
100H
-W <- Write Command(W) ใช้เขียนข้อมูลลง DISK
Writing 00008 bytes
          └── จำนวน Byte ที่เขียนซึ่งจะเท่ากับค่าใน CX
```

เมื่อท่านได้เขียนไฟล์ลงดิสค์แล้ว ถ้าหากว่าท่านต้องการอ่านขึ้นมาก็สามารถทำได้ดังตัวอย่างที่ 9

**ตัวอย่างที่ 9**

```
-N test.bin
-L <-- คำสั่ง Load ไฟล์ ที่มีชื่อตามที่กำหนดในคำสั่ง N(Name)
-U100 <--ใช้ดูว่าเป็นไฟล์ที่ท่านต้องการหรือไม่ ?
   └── Address เริ่มต้นของไฟล์ที่ Load ขึ้นมา
   └── คำสั่งแปลงภาษาเครื่องให้เป็นภาษา Assembly
```

ที่ผ่านมาเป็นการติดต่อกับหน่วยความจำอย่างเดียวยังไม่มีการติดต่อกับอุปกรณ์รอบข้าง ต่อไปนี้เราจะมาเขียนโปรแกรมเพื่อติดต่อจอภาพและแป้นพิมพ์กัน ก่อนอื่นเราต้องทำการศึกษารหัสควบคุมและฟังก์ชันต่าง ๆ ของอินเตอร์รัพต์ 21H กันสักประมาณ 2-3 ฟังก์ชัน ดังตัวอย่างที่ 10

**ตัวอย่างที่ 10**

**รหัสควบคุม**

1. 07H = Bell
2. 0AH = Line Feed (ย้าย Cursor ขึ้นบรรทัดใหม่)
3. 0DH = Carriage Return(ย้าย Cursor มาต้นบรรทัด)

**ฟังก์ชันของดอส บางฟังก์ชันที่จะใช้ในตอนนี้**

**ฟังก์ชัน #1** ใช้รับค่าจากแป้นพิมพ์และแสดงอักขระที่รับเข้ามาออกทางจอภาพ

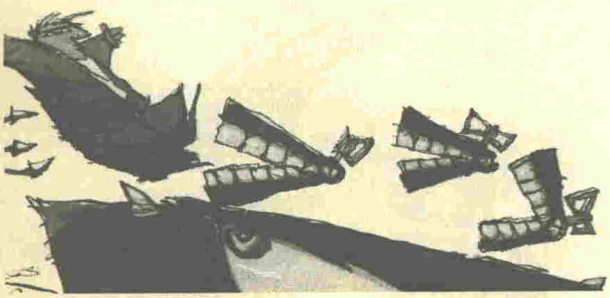
**รูปแบบการใช้** mov ah,01
int 21h
ผลลัพธ์ AL=ตัวอักขระที่รับเข้ามาจากแป้นพิมพ์

**ฟังก์ชัน #2** ใช้ในการแสดงอักขระออกไปที่จอภาพ

**รูปแบบการใช้** mov ah,02
mov dl,อักขระที่ต้องการแสดงผล
int 21h

**ฟังก์ชัน #8** ใช้รับอักขระจากแป้นพิมพ์แต่จะไม่แสดงอักขระที่รับเข้ามาออกทางจอภาพ

**รูปแบบการใช้** mov ah,08
int 21h
ผลลัพธ์ AL=ตัวอักขระที่รับเข้ามาจากแป้นพิมพ์



# โปรแกรมตัวอย่าง

1. โปรแกรมพิมพ์ A-Z ออกไปที่จอภาพเมื่อพิมพ์เสร็จแต่ละตัวต้องขึ้นบรรทัดใหม่และเมื่อพิมพ์เสร็จทั้งหมด 26 ตัวแล้ว ให้ส่งเสียง Bell 1 ครั้ง (ดูตัวอย่างที่ 11)

**ตัวอย่างที่ 11**

```
-a100
54A7:0100 mov cx,1a ; จำนวนรอบที่จะทำงาน 26 รอบ
54A7:0103 mov bl,41 ; อักษร A Ascii Code คือ 41H ท่านสามารถ
หาดูได้จากตา                    ราง ASCII CODE
54A7:0105 mov ah,02 ; ฟังก์ชัน #2 ใช้พิมพ์อักขระออกไปที่จอภาพ
54A7:0107 mov dl,bl ; DL=อักขระที่จะพิมพ์ออกซึ่งอยู่ใน BL
54A7:0109 int 21

54A7:010B mov ah,02
54A7:010D mov dl,0a ; Line Feed ถ้าใช้ Carriage Return แทนจะ
เป็นการ
54A7:010F int 21h  ; พิมพ์ที่ตำแหน่งเดิม

54A7:0111 inc bl   ; เพิ่มค่า BL อีก 1 เหมือนคำสั่ง ADD BL,1
54A7:0113 loop 105  ; Loop ไปที่ Address 105 เลื่อย ๆ จนกว่า
CX=0 โดยที่คำสั่งนี้จะลดค่า CX ลงทีละ 1 โดยอัตโนมัติ
54A7:0115 mov ah,02
54A7:0117 mov dl,07 ; ส่งเสียง Bell 1 ครั้ง
54A7:0119 int 21
54A7:011B
-g=100 11b <-- คำสั่ง Run Program ทดลอง Run ดูนะครับ
   └── Address สุดท้าย
   └── Address เริ่มต้น
   └── GO Command(G) ใช้ RUN PROGRAM
```

2. โปรแกรมการรับค่าจากแป้นพิมพ์ใช้ฟังก์ชัน #8 แล้วพิมพ์ออกจอภาพด้วยฟังก์ชัน #2 (ดูตัวอย่างที่ 12)

**ตัวอย่างที่ 12**

```
-a100
54A7:0100 mov ah,8  ;รับค่าจากแป้นพิมพ์ถ้าใช้ฟังก์ชัน #2 จะ
แสดงผลของ
54A7:0102 int 21   ;ค่าที่รับเข้ามาออกทางจอภาพค่าที่รับเข้ามา
จะอยู่ใน AL

54A7:0104 mov ah,2  ;พิมพ์ค่าที่รับเข้ามาออกไปที่จอภาพด้วย
ฟังก์ชัน #2
54A7:0106 mov dl,al ;DL<--AL ซึ่งคือค่าที่รับเข้ามาทางแป้นพิมพ์
54A7:0108 int 21
54A7:010A
```

3. โปรแกรมอีกประเภทหนึ่งก็คือโปรแกรมที่มีการจัดการกับหน่วยความจำในตำแหน่งที่ต้องการโดยตรง เช่น เราจะเขียนโปรแกรมที่ทำงานเหมือนกับคำสั่ง F 150 15F 41 (ดูตัวอย่างที่ 13)

**ตัวอย่างที่ 13**

```
-a100
54A7:0100 mov cx,10
54A7:0103 mov si,150 ;ให้ค่า SI มาชี้ที่ตำแหน่ง 150H
54A7:0106 mov bl,41  ;BL<-41H
54A7:0108 mov [si],bl;ใส่ค่า BL ในตำแหน่งที่ SI ชี้อยู่เราไม่
สามารถใส่ค่าคงที่ลงไปในหน่วยความจำโดยตรงได้ต้องใส่ผ่าน Register
54A7:010A inc si     ;เพิ่มค่า SI ขึ้นอีก 1
54A7:010B loop 108   ;ทำซ้ำใน Address 108,10A จนกว่า
CX=0
54A7:010D
```

ถ้าต้องการนำค่าใน Address 150-15FH มาแสดงผลที่จอภาพก็สามารถทำได้ดังตัวอย่างที่ 14

**ตัวอย่างที่ 14**

```
-a100
54A7:0100 mov cx,10
54A7:0103 mov si,150  ;ให้ SI ชี้ที่ Address 150H
54A7:0106 mov ah,02   ;ใช้ Function #2 ในการพิมพ์ข้อมูล
ออกจอภาพ
54A7:0108 mov dl,[si] ;DL=ค่าในตำแหน่งที่ SI ชี้อยู่
54A7:010A int 21
54A7:010C inc si
54A7:010D loop 106
54A7:010F
```

ทดลองคีย์โปรแกรมแล้วลองรันดูนะครับจะได้เข้าใจดียิ่งขึ้น

ผมขอจบไว้เพียงเท่านี้นะครับ หวังเป็นอย่างยิ่งว่าคงจะทำให้ท่านหันมาศึกษาภาษาแอสเซมบลี เพื่อที่ท่านเองจะได้มีอัศวินม้าขาวมาไว้ช่วยเหลือยามคับขัน และการศึกษามันก็ไม่ใช่เรื่องยากถ้าท่านศึกษาให้ถูกวิธีและก็ศึกษาอย่างค่อยเป็นค่อยไป เพราะภาษาแอสเซมบลีเป็นภาษาที่มีความละเอียดอ่อนในตัวเป็นอย่างมาก สำหรับตอนต่อไปจะเป็นการใช้แอสเซมบลีในการแปลโปรแกรม การเขียนโปรแกรมจัดการกับจอภาพ การเขียนโปรแกรมติดต่อกับแฟ้มข้อมูล การติดต่อ VEDIO RAM การเขียนกราฟิกและอื่น ๆ อีกมากมาย

พบกับ
Internet
(นิตยสารรายเดือน)
ทุกหน้าเข้มด้วยคุณภาพ
และเนื้อหาสาระ
จะนำคุณเข้าสู่
ความมหัศจรรย์
ของอินเทอร์เน็ต
คุณจะพบว่าโลกของ
ข้อมูล ข่าวสารกว้างไกล
และคุณมีความสามารถที่
จะหยิบมาใช้งานได้
วางแผงแล้ววันนี้
จัดทำโดย
นิตยสาร
MICROCOMPUTER MAGAZINE
ไมโคร
คอมพิวเตอร์
สำหรับผู้สนใจไมโครคอมพิวเตอร์ทุกระดับ